Go Performance Profiling

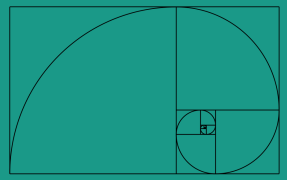
Florian Lehner

Who works on a code base with …

- a test coverage of >50%?

Who works on a code base with …

- a test coverage of >50%?
- a benchmark coverage of >50%?



Fibonacci

```
//go:noinline
func fibonacciRecursive(n uint32) uint32 {
	if n < 2 {
		return n
	}
	return fibonacciRecursive(n-1) + fibonacciRecursive(n-2)
}
```

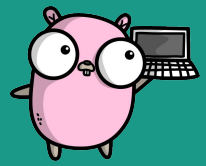
```
//go:noinline
func fibonacciCache(n uint32) uint32 {
	cache := make(map[uint32]uint32, n)
	cache[0] = 0
	cache[1] = 1

	var i uint32
	for i = 2; i <= n; i++ {
		tmp := cache[i-1] + cache[i-2]
		cache[i] = tmp
	}
	return cache[n]
}
```

```
//go:noinline
func fibonacciLoop(n uint32) uint32 {
	var a uint32 = 0
	var b uint32 = 1
	for i := 0; i < int(n); i++ {
		a, b = b, a+b
	}
	return a
}
```

Which function performs best?

- fibonacciRecursive?
- fibonacciCache?
- fibonacciLoop?

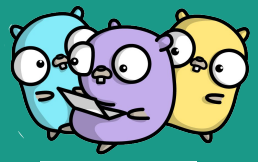


func BenchmarkXXX(testing.B)

```
$ go test -bench=Cache ./...
goos: linux
goarch: amd64
pkg: goperf
cpu: AMD Ryzen 7 PRO 4750U with Radeon Graphics
BenchmarkCache/7-16     6687144        71.3 ns/op         0 B/op         0 allocs/op
BenchmarkCache/17-16     869095        1533 ns/op       330 B/op         1 allocs/op
BenchmarkCache/23-16     466436        2332 ns/op       385 B/op         2 allocs/op
BenchmarkCache/29-16     330007        3041 ns/op       649 B/op         1 allocs/op
BenchmarkCache/43-16     269581        4698 ns/op       734 B/op         3 allocs/op
PASS
ok     goperf      6.138s
```

```
$ go test -bench=Recursive ./...
goos: linux
goarch: amd64
pkg: goperf
cpu: AMD Ryzen 7 PRO 4750U with Radeon Graphics
BenchmarkRecursive/7-16         18015765          65.34 ns/op            0 B/op          0 allocs/op
BenchmarkRecursive/17-16          141086           8062 ns/op            0 B/op          0 allocs/op
BenchmarkRecursive/23-16            7948         150353 ns/op            0 B/op          0 allocs/op
BenchmarkRecursive/29-16             450        2534529 ns/op            0 B/op          0 allocs/op
BenchmarkRecursive/43-16               1     3159219012 ns/op            0 B/op          0 allocs/op
PASS
ok      goperf      10.151s
```

```
$ go test -bench=Loop ./...
goos: linux
goarch: amd64
pkg: goperf
cpu: AMD Ryzen 7 PRO 4750U with Radeon Graphics
BenchmarkLoop/7-16      302948274        3.795 ns/op       0 B/op       0 allocs/op
BenchmarkLoop/17-16     182696710        6.763 ns/op       0 B/op       0 allocs/op
BenchmarkLoop/23-16     134103073        8.261 ns/op       0 B/op       0 allocs/op
BenchmarkLoop/29-16     111308197        10.11 ns/op       0 B/op       0 allocs/op
BenchmarkLoop/43-16     77693318         14.44 ns/op       0 B/op       0 allocs/op
PASS
ok     goperf      8.130s
```



```
import _ "net/http/pprof"
```

```
import     _ "net/http/pprof"

[...]
        for {
                for num, expected := range numbers {
                        if result = fibonacciCache(num); result != expected {
                                panic(fmt.Sprintf("%d != %d", result, expected))
                        }
                        if result = fibonacciRecursive(num); result != expected {
                                panic(fmt.Sprintf("%d != %d", result, expected))
                        }
                        if result = fibonacciLoop(num); result != expected {
                                panic(fmt.Sprintf("%d != %d", result, expected))
                        }
                }
        }
```

```
$ curl -o profile.out http://localhost:6060/debug/pprof/profile?seconds=10

$ go tool pprof -http=:8080 profile.out
Serving web UI on http://localhost:8080
```

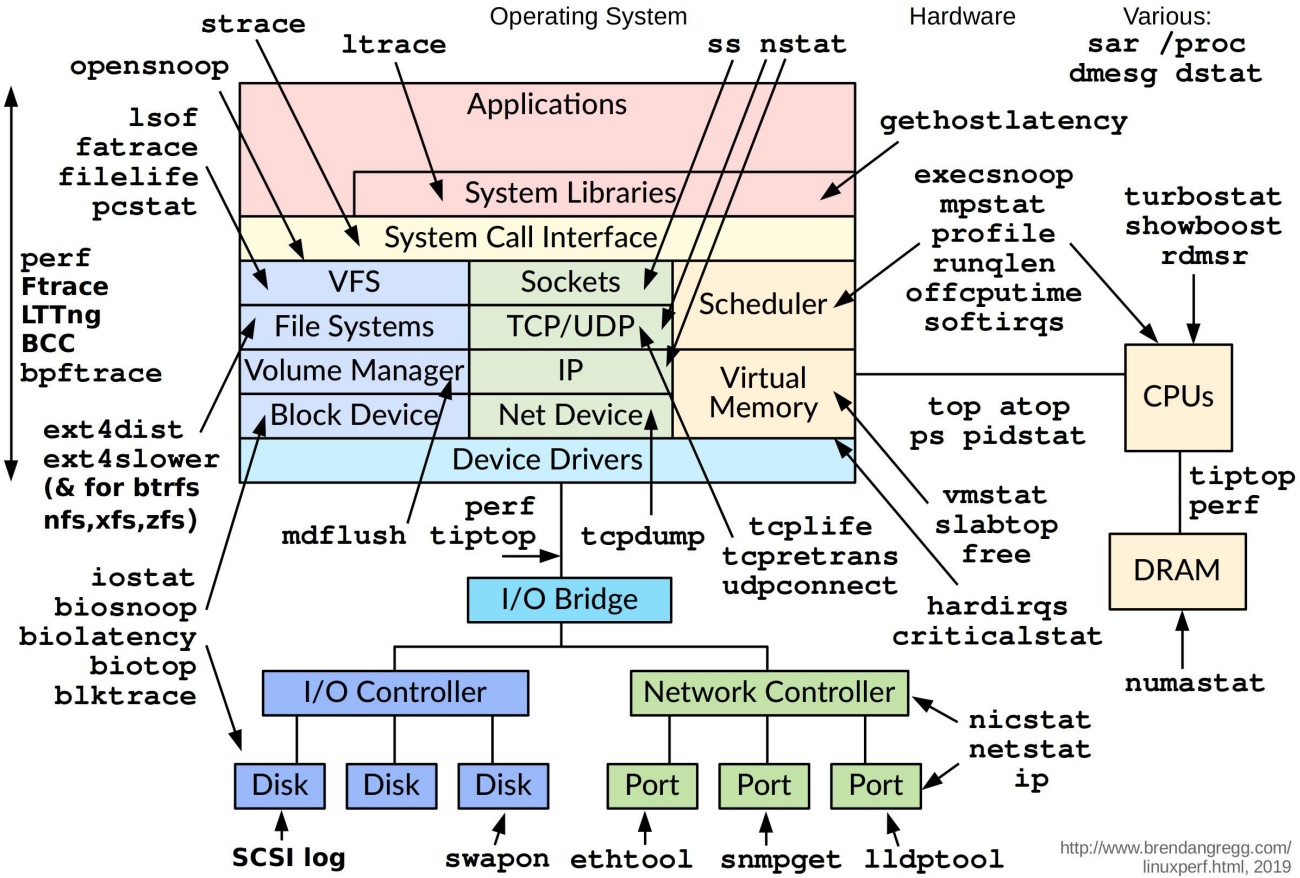

pprof
VIEW
SAMPLE
REFINE
CONFIG
DOWNLOAD
Search regexp
goperf cpu
root
runtime.main
main.main
main.fibonacciRecursive
main.fibo…

Flat	Flat%	Sum%	Cum	Cum%	Name
999	99.90%	99.90%	999	99.90%	main.fibonacciRecursive
0	0.00%	99.90%	999	99.90%	runtime.main
0	0.00%	99.90%	999	99.90%	main.main

File: goperf
Type: samples
Time: Nov 13, 2022 at 7:42pm (CET)
Duration: 10.13s, Total samples = 1000
Showing nodes accounting for 999, 99.90% of 1000 total
Dropped 1 node (cum <= 5)

perf

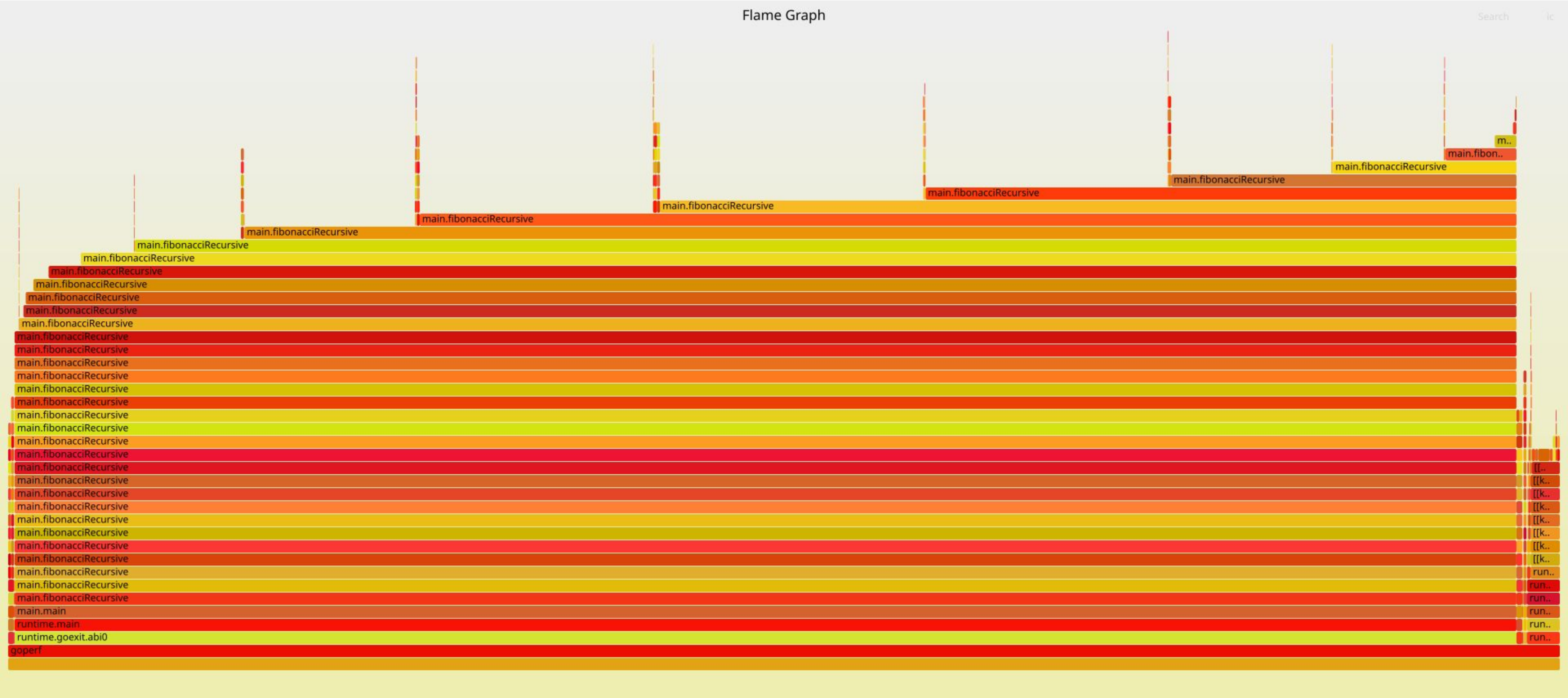
Linux Performance Observability Tools



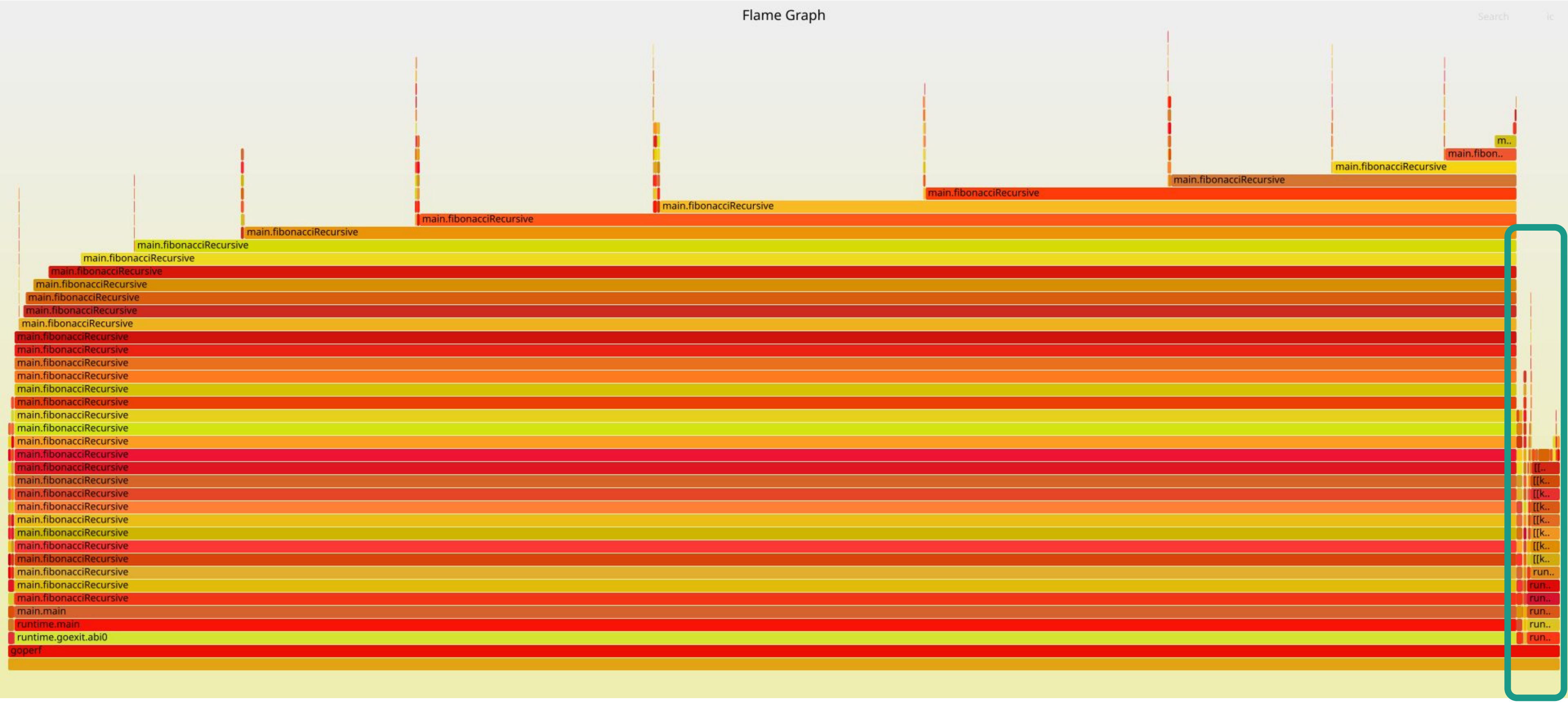
http://www.brendangregg.com/linuxperf.html, 2019

http://www.brendangregg.com/linuxperf.html

```
$ perf record --call-graph dwarf -F 257 -p $(pgrep goperf) -- sleep 10
$ perf script -F comm,pid,tid,time,event,ip,sym,dso,trace > out.perf
$ stackcollapse-perf.pl --all --context --addrs --inline out.perf > out.folded
$ flamegraph.pl out.folded > goperf.svg
```



Flame Graph
Search
main.fibonacciRecursive
main.fibon..
m..
main.main
runtime.main
runtime.goexit.abi0
goperf


Flame Graph
Search
main.fibonacciRecursive
main.main
runtime.main
runtime.goexit.abi0
goperf

```
$ perf record --call-graph dwarf -F 257 -p $(pgrep goperf) -- sleep 10
$ perf script -F comm,pid,tid,time,event,ip,sym,dso,trace > out.perf
$ stackcollapse-perf.pl --all --context --addrs --inline out.perf > out.folded
$ flamegraph.pl out.folded > goperf.svg
```

eBPF/uprobes

```
$  bpftrace -l 'uprobe:goperf.test:*'
uprobe:goperf.test:goperf.fibonacciCache
uprobe:goperf.test:goperf.fibonacciLoop
uprobe:goperf.test:goperf.fibonacciRecursive

[...] ✂️

uprobe:goperf.test:main.main
uprobe:goperf.test:runtime.gcDrain
uprobe:goperf.test:runtime.mallocgc
uprobe:goperf.test:runtime.mallocinit
```

U(ret)probes are not for free

```
$ go test -c ./...
$ goperf.test -test.bench=. -test.count=10 > without_uprobes.txt
$ bpftrace -e 'uprobe:goperf.test:goperf.fibonacci* { printf("%s\n", func); }'
$ goperf.test -test.bench=. -test.count=10 > with_uprobes.txt
$ benchstat without_uprobes.txt with_uprobes.txt
```

```
$ go test -c ./....
$ goperf.test -test.bench=. -test.count=10 > without_uprobes.txt
$ bpftrace -e 'uprobe:goperf.test:goperf.fibonacci* { printf("%s\n", func); }'
$ goperf.test -test.bench=. -test.count=10 > with_uprobes.txt
$ benchstat without_uprobes.txt with_uprobes.txt
name              old time/op    new time/op      delta
Loop/7-16          3.82ns ± 4%   648.13ns ± 1%   +16859.29%  (p=0.000 n=10+9)
Loop/17-16         6.91ns ± 2%   633.86ns ± 4%    +9078.69%  (p=0.000 n=9+10)
Loop/23-16         8.31ns ± 4%   652.13ns ± 3%    +7746.97%  (p=0.000 n=10+10)
Cache/7-16          177ns ± 3%     841ns ± 3%     +376.03%  (p=0.000 n=10+10)
Cache/17-16        2.02µs ± 3%    3.30µs ± 9%      +63.67%  (p=0.000 n=10+10)
Cache/23-16        2.96µs ± 7%    4.10µs ±12%      +38.79%  (p=0.000 n=10+10)
Recursive/7-16     65.2ns ± 3%  26241.7ns ± 3%   +40128.26%  (p=0.000 n=10+10)
Recursive/17-16    7.95µs ± 5%  3340.73µs ± 4%   +41911.18%  (p=0.000 n=10+9)
Recursive/23-16     146µs ± 3%    59030µs ± 5%    +40278.67%  (p=0.000 n=9+10)
```

```
#!/usr/bin/env bpftrace

uprobe:goperf.test:goperf.fibonacci*
{
        @start[tid] = nsecs;
}

uretprobe:goperf.test:goperf.fibonacci*
/@start[tid]/
{
        @usecs = hist((nsecs - @start[tid]) / 1000);
        delete(@start[tid]);
}

END
{
        clear(@start);
}
```

```
runtime: g 52: unexpected return pc for goperf.fibonacciCache called from 0x7ffffffe000
stack: frame={sp:0xc000071df0, fp:0xc000071eb0} stack=[0xc000071000,0xc000072000)

0x000000c000071de0:    0x000000c000071ea0   0x00000000005c205d
0x000000c000071df0: <  0x000000c00009be08  0x000001000041b366
0x000000c000071e00:    0x00007fb4773ec2b0    0x0000000000000340
0x000000c000071e10:    0x00007fb4773e3108    0x0000000000000380
0x000000c000071e20:    0x000000c000280c00   0x000000c000018700
0x000000c000071e30:    0x0000000000000000  0x0000010000000001
0x000000c000071e40:    0x000000c00009be68  0x000000000040efb2
0x000000c000071e50:    0x0000000000000340  0x00000000005febc0
0x000000c000071e60:    0x000000c000280c01   0x000000c00009bea8
0x000000c000071e70:    0x0000000000000000  0x5ca46bd300000200
0x000000c000071e80:    0x0000000000000000  0x0000000000000000
0x000000c000071e90:    0x0000000000000000  0x0000000000000000
0x000000c000071ea0:    0x000000c00009bf18   !0x00007ffffffe000
0x000000c000071eb0: > 0x0000000000000017  0x000000c00009cea0
```

```
$ objdump -t goperf  | grep fibonacci
00000000005c1fc0 g     F .text    000000000000017f          goperf.fibonacciCache
00000000005c2140 g     F .text    000000000000001f          goperf.fibonacciLoop
00000000005c2160 g     F .text    0000000000000064          goperf.fibonacciRecursive

(gdb) disassemble 0x5c1fc0
Dump of assembler code for function goperf.fibonacciCache:
=> 0x00000000005c1fc0 <+0>:     lea    -0x38(%rsp),%r12
   0x00000000005c1fc5 <+5>:     cmp    0x10(%r14),%r12
   0x00000000005c1fc9 <+9>:     jbe    0x5c212d <goperf.fibonacciCache+365>
[..]
   0x00000000005c211b <+347>:   mov    (%rax),%eax
   0x00000000005c211d <+349>:   mov    0xb0(%rsp),%rbp
   0x00000000005c2125 <+357>:   add    $0xb8,%rsp
   0x00000000005c212c <+364>:   ret
   0x00000000005c212d <+365>:   mov    %eax,0x8(%rsp)
   0x00000000005c2131 <+369>:   call   0x469100 <runtime.morestack_noctxt>
   0x00000000005c2136 <+374>:   mov    0x8(%rsp),%eax
   0x00000000005c213a <+378>:   jmp    0x5c1fc0 <goperf.fibonacciCache>
```

Go runtime specific to grow the stack

Questions?

Slides
https://github.com/florianl/talks

Gophers by
github.com/ashleymcnamara/gophers


https://www.pexels.com/photo/blur-book-stack-books-bookshelves-590493/